CSc 120
Introduction to Computer Programming II

Adapted from slides by
Dr. Saumya Debray

01-d: Python review
python review: tuples
Tuples

>>> x = (111, 222, 333, 444, 555)
>>> x
(111, 222, 333, 444, 555)
>>> x[0]
111
>>> x[2]
333
>>> x[-1]
555
>>> x[-2]
444

a tuple is a sequence of values (like lists)
Tuples

>>> x = (111, 222, 333, 444, 555)
>>> x
(111, 222, 333, 444, 555)
>>> x[0]
111
>>> x[2]
333
>>> x[-1]
555
>>> x[-2]
444

a tuple is a sequence of values (like lists)

tuples use parens ()
  • by contrast, lists use square brackets []
    • parens can be omitted if no confusion is possible
  • special cases for tuples:
    • empty tuple: ()
    • single-element tuple: must have comma after the element:
      (111,)
Tuples

a tuple is a sequence of values (like lists)

tuples use parens ()
• by contrast, lists use square brackets []
  • parens can be omitted if no confusion is possible
• special cases for tuples:
  • empty tuple: ()
  • single-element tuple: must have comma after the element:

    (111,)

indexing in tuples works similarly to strings and lists

>>> x = (111, 222, 333, 444, 555)
>>> x
(111, 222, 333, 444, 555)
>>> x[0]
111
>>> x[2]
333
>>> x[-1]
555
>>> x[-2]
444
>>>
Tuples

>>> x = (111, 222, 333, 444, 555)

computing a length of a tuple: similar to strings and lists

len(x)
5

>>> x[2:]
(333, 444, 555)

>>> x[:4]
(111, 222, 333, 444)

>>> x[1:4]
(222, 333, 444)

>>>
Tuples

>>> x = (111, 222, 333, 444, 555)

>>> len(x)
5

>>> x[2:]
(333, 444, 555)

>>> x[:4]
(111, 222, 333, 444)

>>> x[1:4]
(222, 333, 444)
Tuples

>>> x = (111, 222, 333, 444, 555)
>>> x
(111, 222, 333, 444, 555)

>>> y = (666, 777, 888)

>>> x + y
(111, 222, 333, 444, 555, 666, 777, 888)

>>> y * 3
(666, 777, 888, 666, 777, 888, 666, 777, 888)

+ and * work similarly on tuples as for lists and strings
Tuples

```python
>>> x = (111, 222, 333, 444, 555)

>>> for item in x:
    print(item)

111
222
333
444
555

>>> 222 in x
True

>>> 999 in x
False
```
Tuples

>>> x = (111, 222, 333, 444, 555)
>>> for item in x:
    print(item)
111
222
333
444
555
>>> 222 in x
True
>>> 999 in x
False

iterating through the elements of a tuple: similar to lists and strings

checking membership in a tuple: similar to lists and strings
Tuples

>>> x = (111, 222, 333, 444, 555)
>>> x
(111, 222, 333, 444, 555)
>> x[2]
333
>>> x[2] = 999
Traceback (most recent call last):
  File "<pyshell#102>", line 1, in <module>
    x[2] = 999
TypeError: 'tuple' object does not support item assignment

Tuples are not mutable
Sequence types: mutability

tuples are immutable

```python
>>> x = ( ['aa', 'bb'], ['cc', 'dd'], ['ee'] )
>>> x[0] = 'ff'
Traceback (most recent call last):
  File "<pyshell#108>", line 1, in <module>
    x[0] = 'ff'
TypeError: 'tuple' object does not support item assignment
```
Sequence types: mutability

```python
>>> x = ( ['aa', 'bb'], ['cc', 'dd'], ['ee'] )
>>> x[0] = 'ff'
Traceback (most recent call last):
  File "<pyshell#108>", line 1, in <module>
    x[0] = 'ff'
TypeError: 'tuple' object does not support item assignment

>>> x[0][0] = 'ff'
```

Lists are mutable:

```python
>>> x
(['ff', 'bb'], ['cc', 'dd'], ['ee'])
```
Sequence types: mutability

>>> x = ( ['aa', 'bb'], ['cc', 'dd'], ['ee'] )

>>> x[0] = 'ff'  

`tuple` object does not support item assignment

>>> x[0][0] = 'ff'

`str` object does not support item assignment

`list` object is mutable

>>> x[0][0][0] = 'a'

`tuple` object is immutable
Sequence types: mutability

tuple (immutable)

list (mutable)

string (immutable)
Sequence types: mutability
EXERCISE

>>> x = [ (1, 2, 3), (4, 5, 6), (7, 8, 9) ]

>>> x[0][0] = (2, 3, 4)

what do you think will be printed out?

>>> x[0] = [ 2, 3, 4 ]

what do you think will be printed out?
Why use tuples?

At the implementation level, tuples are much simpler than lists:

• lists are mutable; tuples are immutable
  • this means that the implementation can process tuples without having to worry about the possibility of updates

• lists have methods (e.g., append); tuples do not have methods

⇒ Tuples can be implemented more efficiently than lists
Summary: sequence types

Sequence types include: strings, lists, and tuples

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x in s</code></td>
<td>True if an item of s is equal to x, else False</td>
</tr>
<tr>
<td><code>x not in s</code></td>
<td>False if an item of s is equal to x, else True</td>
</tr>
<tr>
<td><code>s + t</code></td>
<td>the concatenation of s and t</td>
</tr>
<tr>
<td><code>s * n</code> or <code>n * s</code></td>
<td>equivalent to adding s to itself n times</td>
</tr>
<tr>
<td><code>s[i]</code></td>
<td><code>i</code>th item of s, origin 0</td>
</tr>
<tr>
<td><code>s[i:j]</code></td>
<td>slice of s from <code>i</code> to <code>j</code></td>
</tr>
<tr>
<td><code>s[i:j:k]</code></td>
<td>slice of s from <code>i</code> to <code>j</code> with step <code>k</code></td>
</tr>
<tr>
<td><code>len(s)</code></td>
<td>length of s</td>
</tr>
<tr>
<td><code>min(s)</code></td>
<td>smallest item of s</td>
</tr>
<tr>
<td><code>max(s)</code></td>
<td>largest item of s</td>
</tr>
<tr>
<td><code>s.index(x[, i[, j]])</code></td>
<td>index of the first occurrence of x in s (at or after index <code>i</code> and before index <code>j</code>)</td>
</tr>
<tr>
<td><code>s.count(x)</code></td>
<td>total number of occurrences of x in s</td>
</tr>
</tbody>
</table>

The elements are: $i$, $i+k$, $i+2k$, ...

Source: https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range
python review: dictionaries
Dictionaries

• A dictionary is like an array, but it can be indexed using strings (or numbers, or tuples, or any immutable type)
  • the values used as indexes for a particular dictionary are called its keys
  • think of a dictionary as an unordered collection of key : value pairs
  • empty dictionary: {}

• It is an error to index into a dictionary using a non-existent key
Dictionaries

>>> crs_units = {}
empty dictionary

>>> crs_units['csc 110'] = 4

>>> crs_units['csc 120'] = 4

>>> crs_units['csc 352'] = 3

>>> course = 'csc 110'

>>> crs_units[course]
4

>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}

>>>
Dictionaries

```python
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>> course = 'csc 110'

>>> crs_units[course]
4

>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
```
Dictionaries

>>> crs_units = {}
empty dictionary

>>> crs_units['csc 110'] = 4
populating the dictionary

>>> crs_units['csc 120'] = 4

>>> crs_units['csc 352'] = 3

>>> course = 'csc 110'

>>> crs_units[course]
looking using keys (indexing)

4

>>> crs_units

{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}

>>>
Dictionaries

>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>> course = 'csc 110'

>>> crs_units[course]
4

>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}

empty dictionary

populating the dictionary
• in this example, one item at a time

looking using keys (indexing)
• we can populate it using this syntax
Dictionaries

```python
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>> course = 'csc 110'
>>> crs_units[course]
4
>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
```  
empty dictionary

populating the dictionary
- in this example, one item at a time

looking using keys (indexing)
- we can populate it using this syntax

```python
>>> crs_units['mis 115']
Traceback (most recent call last):
  File "<pyshell#12>" , line 1, in <module>
    crs_units['mis 115']
KeyError: 'mis 115'
```  
indexing with a key not in the dictionary is an error ( `KeyError` )
Dictionaries

```python
>>> crs_units = {}
>>> crs_units['csc 110'] = 4
>>> crs_units['csc 120'] = 4
>>> crs_units['csc 352'] = 3
>>> course = 'csc 110'

>>> crs_units
{'csc 110': 4, 'csc 120': 4, 'csc 352': 3}
```

Indexing with a key not in the dictionary is an error (KeyError)

```
>>> 'mis 115' in crs_units
False
```

In operator:
- returns True if a key is in the dictionary an False otherwise
Dictionaries

>>> crs_units = { 'csc 110': 4, 'csc 120': 4, 'csc 352': 3 }

>>> crs_units['csc 110']
4

>>> crs_units.keys()
dict_keys(['csc 110', 'csc 120', 'csc 352'])

>>> crs_units.values()
dict_values([4, 4, 3])
Dictionaries

>>> crs_units = { 'csc 110': 4, 'csc 120': 4, 'csc 352': 3 }

>>> crs_units['csc 110']
4

>>> crs_units.keys()
dict_keys(['csc 110', 'csc 120', 'csc 352'])

>>> crs_units.values()
dict_values([4, 4, 3])

methods for getting:
• the keys
• the values
Dictionaries

```python
>>> crs_units = { 'csc 110': 4, 'csc 120': 4, 'csc 352': 3}

>>> crs_unist['csc 110']
4

>>> crs_units.keys()
dict_keys(['csc 110', 'csc 120', 'csc 352'])

>>> list(crs_units.keys())
['csc 110', 'csc 120', 'csc 352']
```

get a list of the keys
Dictionaries

>>> crs_units = { 'csc 110': 4, 'csc 120': 4, 'csc 352': 3}

>>> for crs in crs_units:  # We can use a for loop to iterate through a dictionary
    print(crs, crs_units[crs])

  csc 110 4
  csc 352 3
  csc 120 4
Dictionaries

```python
>>> crs_units = { 'csc 110': 4, 'csc 120': 4, 'csc 352': 3 }

>>> for crs in crs_units:
    print(crs + " : ", crs_units[crs], " units")

csc 110 : 4 units

csc 352 : 3 units

csc 120 : 4 units
```

We can use a `for` loop to iterate through a dictionary.

Note: this may not list the items in the dictionary in the same order as when they were inserted.
Dictionaries

```python
>>> crs_units = {'csc 110': 4, 'csc 120': 4, 'csc 352': 3}

>>> for crs in crs_units:
    # print(crs + ":", crs_units[crs], "units")
    print("{0}:{1} units".format(crs, crs_units[crs]))

csc 110: 4 units
csc 352: 3 units
csc 120: 4 units
```
EXERCISE

```python
>>> crs_units = { 'csc 352' : 3, 'csc 120': 4, 'csc 110': 4 }
>>> for crs in crs_units:
    print( "{0} : {1} units".format( crs, crs_units[crs] ) )

  csc 110 : 4 units
  csc 120 : 4 units
  csc 352 : 3 units

How can we get the dictionary contents to be printed out in sorted order of the keys?
(I.e., what goes in the box?)
```
EXERCISE

Write a function `count_chars(s)` that takes a string `s` and returns a dictionary of the counts of all characters in the string.
Write a function `count_chars(s)` that takes a string `s` and returns a dictionary of the counts of all characters in the string.

```python
def count_chars(s):
    counts = {}
    s = s.lower()
    for c in s:
        if c in counts:  # if we have seen c, increment its count
            counts[c] = counts[c] + 1
        else:             # otherwise, it is the first occurrence
            counts[c] = 1
    return counts
```
## Dictionary Summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dict()</code></td>
<td>Return an empty dictionary.</td>
</tr>
<tr>
<td><code>len(d)</code></td>
<td>Return the number of items in the dictionary <code>d</code>.</td>
</tr>
<tr>
<td><code>d[key]</code></td>
<td>Return the item of <code>d</code> with key <code>key</code>. Raises an error if <code>key</code> is not in the dictionary.</td>
</tr>
<tr>
<td><code>d[key] = value</code></td>
<td>Set <code>d[key]</code> to <code>value</code>.</td>
</tr>
<tr>
<td><code>del d[key]</code></td>
<td>Remove <code>d[key]</code> from <code>d</code>. Raises an error if <code>key</code> is not in the dictionary.</td>
</tr>
<tr>
<td><code>key in d</code></td>
<td>Return <code>True</code> in <code>d</code> has a key <code>key</code>, else <code>False</code>.</td>
</tr>
<tr>
<td><code>key not in d</code></td>
<td>Equivalent to not key in <code>d</code>.</td>
</tr>
<tr>
<td><code>keys()</code></td>
<td>Return the dictionary's keys.</td>
</tr>
<tr>
<td><code>values()</code></td>
<td>Return the dictionary's values.</td>
</tr>
<tr>
<td><code>items()</code></td>
<td>Return the dictionary's items as tuples.</td>
</tr>
</tbody>
</table>